Towards a platform for empirical software design studies

Koen Yskout
imec-DistriNet, KU Leuven
B-3001 Leuven, Belgium
koen.yskout@cs.kuleuven.be

Dimitri Van Landuyt
imec-DistriNet, KU Leuven
B-3001 Leuven, Belgium
dimitri.vanlanduyt@cs.kuleuven.be

Wouter Joosen
imec-DistriNet, KU Leuven
B-3001 Leuven, Belgium
wouter.joosen@cs.kuleuven.be

***Abstract*—The process of empirical research is founded on careful study design, sound instantiation and planning of the study, and the systematic collection and processing of data. These activities require extensive expertise and know-how, are repetitive, laborious and error-prone, and adequate tool support is currently lacking, particularly in support of empirical software engineering research.**

In this paper, we outline our vision of an integrated end-to-end tool platform that supports these activities and we elaborate on what it would take for such a platform to become a (re)usable platform for the research community.

I. Introduction

The importance of experimental research in Software Engineering can not be overstressed. Empirical studies are essential for the maturation of the discipline and to 'build a science of computer science' [1]. Nevertheless, conducting an empirical software engineering experiment is no walk in the park, especially when the software engineers themselves (rather than just the artifacts they have produced) are part of the study. To ascertain scientific soundness, many guidelines and attention points must be taken into account [2], [3] and quality assurance is a constant concern throughout the many activities that constitute the process of an empirical study [4], [5], in particular (i) scoping, (ii) planning, (iii) operation, (iv) analysis and interpretation, (v) presentation and packaging, and finally (ideally), (vi) study replication.

Many tools exist that can assist the empirical software engineering researcher, but these tools are focused on specific tasks, e.g. the use of R [6] for statistical analysis and hypothesis testing, Hackystat [7] for obtaining software development process data, versioning systems for tracking the evolution of software over time, etc. In the area of product and process engineering, there are several tools that support the design and analysis of experiments (e.g., Design Expert [8] or SAS JMP [9]). These tools are typically not focused on software engineering experiments with human participants, through.

Although these existing tools address specific needs at specific phases in the execution of an empirical study, there is a lack of integrated tooling and systematic end-to-end process support for conducting empirical software engineering experiments. Currently, the necessary tools are created often for the specific purpose of single study, e.g. [10]–[14], and these tools are labor-intensive to create, test, maintain, evolve and reuse.

In this paper, we present our vision towards an integrated tooling platform for conducting experiments that involve software engineering tasks and activities and human participants (software designers, requirement engineers, developers, etc). More specifically, we present a detailed discussion of the benefits such a platform would bring in the different phases of an empirical study [4], in terms of supporting (i) the study design, (ii) study instantiation, (iii) the integration and automation of measurement collection, and we argue how such a platform would contribute to reducing threats to validity and improving reproducibility and quality of documentation. We particularly highlight the potential of integrating best practices and community standards on how to design and conduct software engineering experiments [2]–[5].

This paper is motivated from our experiences from the past five years with performing empirical studies specifically related to early security and privacy design activities and the creation of custom tools to support these studies. These studies involved, for example, the evaluation of using security patterns in architectural design [11], the effects of systematically structuring a catalog of such security patterns [10], and evaluating a privacy threat elicitation and analysis technique [15]. Some of the challenges that we have encountered repeatedly throughout these studies, and for which we expect assistance from the envisioned tooling platform, are (i) defining precise research hypotheses in the area of early software design stages; (ii) translating these hypotheses to an appropriate study design and statistical processing, while avoiding common threats to validity; (iii) dealing with a relatively large number of participants that are expected to concurrently adhere to a complex flow of open-ended architectural design tasks over a long period of time, while still being closely monitored; (iv) obtaining accurate and meaningful measurements in an automated fashion; (v) managing and processing the obtained data in a coherent and repeatable fashion; and (vi) ensuring the privacy of the participants throughout the process.

II. A platform for empirical software design studies

To provide support to the experimenter throughout the entire experimental process, we envision an integrated platform with four components that align to the different phases of the experimental process: (1) a design workbench, in which the

experimenter can plan and design a study; (2) an instantiation tool, in which the designed study is instantiated for a particular execution (e.g., by providing subject data and task descriptions); (3) an execution environment, which drives and controls the experiment, and integrates with software engineering tools to produce the required data; and (4) a processing and packaging tool to analyze, wrap up and share all information about the experiment.

This vision is depicted graphically in Figure 1. Note that, while the figure may give the impression of a strictly linear process for conducting empirical studies, this is almost never the case in reality. Often, a single research question translates into multiple related studies, for example by first conducting a small exploratory study that is used to set expectations and refine hypotheses and designs and test study material, followed by a full-blown study afterwards. Even within a single study, and especially during design and instantiation, iterative refinements may be needed. In all of these cases, an integrated and streamlined platform has the potential to reduce the costs (in terms of time and effort) for setting up and carrying out such studies.

The remainder of this section elaborates further on each of the platform components.

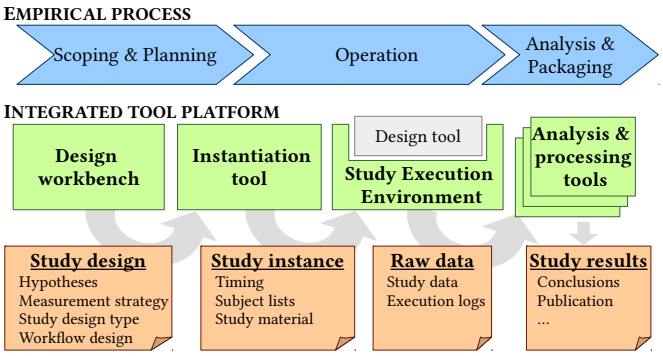

Fig. 1. Overview of the proposed tool platform for empirical software design studies.

A. Design workbench

The design workbench provides support for defining and planning an experiment. Although this activity is highly specific for each experiment, experiments share many similarities. In addition to more systematically recording the decisions of the experimenter, the design workbench is a prime candidate to leverage reusable knowledge and expertise, for example by incorporating templates, making suggestions and recommendations, and giving feedback.

1) Scoping: The scoping step of the experiment clarifies the motivation for the experiment. A platform can help the experimenter by offering the possibility to explicitly enter and store the goals and objectives of the study, and providing templates to do so, for example based on the GQM method [16]. To allow the researcher to demonstrate that the goals (and later the hypotheses) of the study were actually defined before collecting the data, and have not changed since, the platform could integrate with a trusted time stamping authority (a digital notary).

2) Formulating hypotheses: The formulation of the (null and alternative) hypotheses is crucial for a study. They determine the measurements that have to be collected, and how they will be compared. Often, hypotheses for empirical experiments have a similar shape (e.g., a comparison of a mean value between two groups), and so the workbench could offer template hypotheses, perhaps even specialized for certain objectives and experiment designs. There could for example be a hypothesis template for comparing the average performance in terms of required time of two groups that are treated differently (e.g., each group uses a different tool). Such templates can be enriched with suggestions for suitable statistical tests, and inform about the assumptions under which these tests are valid.

3) Study design: Empirical studies are often structured according to specific design types [4], [5], [17]. An experimenter workbench can therefore support the selection of (or even suggest) an adequate design type (such as a completely randomized design, a paired comparison design, or a 2×2 factorial design [4]) and support the design and concretization of the study in line with the selected design type. Suggestions can be based on the information entered by the researcher, such as the considered factors and their levels, the considered variables and the values that they can take, but also which variables are expected to affect the measured quantity, or for which variables the differences across different values are of interest, for example.

4) Workflow design: As part of the experiment design, the experimenter should define the workflow that participants will follow. This can be rather straightforward, for instance when the participant only needs to execute a single task, but it may also be more specific to the design of the study. For example, in an earlier study [11], the adopted workflow forced participants to first complete an entry questionnaire, then to perform a warm-up task, then to execute a set of tasks in a pre-defined order with one technique, and finally to perform another set of tasks using an alternative technique.

The design workbench can support defining an explicit experiment workflow, and keeping it in line with the design of the study. Such a workflow resembles an executable business process, and could make use of concepts and notations from existing business process modeling languages (e.g., BPEL or BPMN), or be based on the work on e-Science workflows [18], [19].

This activity should furthermore trigger the designer to consider and specify exceptional situations, for example by concretizing the overall policy in terms of allowing participants to restart, pause or skip activities during the execution of the experiment and contemplating the effects of such policies on the overall experiment.

5) Threats to validity: As with many aspects of study design, the applicable threats to validity are often similar in shape from one study to another. A workbench can provide

a checklist of generally-applicable threats to validity within the categories of conclusion, construct, internal and external validity [4]. By injecting study-specific information about the goals and hypotheses into this generic checklist, the design workbench can present these threats in a more concrete and understandable way to the experimenter.

6) Privacy and ethics: Experiment that involve human participants inherently may be susceptible to privacy-related and ethical issues, and usually require informed consent of the participants[1]. The design workbench can provide explicit support for this, by aiding to decide how the collected data will be treated (identifiable, pseudonymous or completely anonymous, for example), and determining up front where and how long the data will be stored (data retention).

B. Instantiation tool

The instantiation of an experiment bridges the planning phase to the operational phase. While instantiation is usually considered to be part of either planning or operation, explicit distinction is useful in light of replications, which in fact are multiple instances of the same experiment design that use different subjects, tasks, or software applications to work on. The inclusion of an *instantiation tool* in the platform can complement the study design produced with the design workbench with concrete information on timing, subjects and tasks, to prepare it for execution.

1) Subject selection: Selecting subjects for a study can happen in different ways, for example by random sampling (with or without stratification), or by convenience sampling. The instantiation tool can support the experimenter with carrying out this step; for example, it can implement various sampling algorithms that work on a given list of participants. When stratification is used, the list of subjects should include additional subject attributes of relevance to the study, for example, years of experience, or educational background, and the designer needs to specify the target distribution over these variables. The result is the set of subjects with which the experiment will be carried out. Furthermore, depending on the privacy requirements, the instantiation tool can also be used to generate and assign identifiers (pseudonyms) to the subjects.

2) Create study material: During instantiation, the experiment materials have to be created. This includes preparing the instructions for the participants, descriptions of the tasks that the participants will be asked to execute, any required artifacts (e.g., a design description of the system under study), as well as questionnaires to obtain feedback, for example. The instantiation tool can support this activity, for example by organizing this content into a generic folder structure, such that this information is easy to retrieve, and the experimenter can keep an overview of what is still missing at a glance. Going further, if the instantiation tool is well integrated with the execution environment (as described in the next section), the material can be injected directly into the execution environment, for example by generating configuration files or web pages that reflect the material.

3) Randomization: Depending on the chosen experiment design type, certain aspects of the experiment may need to be randomized. This includes the assignment of the participants to a treatment or control group, or the order in which the different tasks need to be executed by each participant to counter learning effects. Randomization can become more complex, for example when groups need to be balanced, and when a blocked design is used. An instantiation tool can implement this procedure generically, and apply it to the obtained lists of subjects, tasks, etc., such that the experimenter no longer has to do this manually.

C. Study execution environment

Subjects of software engineering studies often use one or more tools to perform their work. Therefore, next to the design workbench that directly supports the experimenter, the platform should also offer (or integrate with) the environment that the participants of the study actually use in practice, for the reasons described below.

1) Development tool integration and interoperability: The execution environment is ideally integrated tightly with the tool(s) that the participant uses to perform the tasks (e.g., a UML or CASE tool, a code editor, or an analysis tool), so that the participants can keep working within the environment they are familiar with. This is important, because the confrontation with a new environment may impact the participants' performance and productivity, and measurements may not reflect real-world conditions. Such integration for the purposes of enacting control and monitoring is however not always feasible. And alternative option is to integrate the participant tool and execution environment only partially, or even run them side by side.

The most basic form of an execution environment could be a website that guides the participants through the experiment workflow, but this inherently limits the capabilities of the tools that can be used (interacting with a UML design tool, for example). In contrast, full integration can be accomplished with an instrumented version of the code editor that the participants commonly use, which automates the complete workflow of the experiment, including data collection and submission.

2) Workflow execution: Even when the experiment design contains a precisely defined workflow, enforcing participants to adhere to this workflow remains difficult. For example, suppose that participants have to hand in a solution for their current task before being given the next task. When enforcing this procedure manually, making sure that every participant receives the right instructions at the right time is cumbersome and error-prone, especially with larger groups of participants that work simultaneously and when task orders are randomized. The execution environment of the experiment can make this more practical and uniform, by guiding the participant through the process, thereby enforcing the participant to adhere to the intended workflow. It thus needs to have the

[1]While the potential harm for the participants in the context of software engineering experiments is often limited, the experimenter should still consider these issues from the start.

ability to control and monitor workflow execution, display the right information at the right time, and ideally integrate or instrument the development tools involved in the study.

Automation is the main benefit of using such an execution environment, in the sense that it becomes feasible for participants to work without supervision of the experimenter, which greatly reduces the resources that are required for performing a larger scale experiments. Furthermore, persisting the workflow state for resuming it later facilitates experiments that extend over longer periods of time.

3) Data collection: Accurate data is of paramount importance for the validity of an experiment. Unfortunately, obtaining high-quality data in software engineering experiments with human participants is difficult. Self-reporting is often imprecise or unreliable, because participants forget what they have done or how long it took, or may have an incentive to be untruthful about this. Manual observation by the experimenter accommodates these concerns, but it does not scale well to experiments that involve a large number of participants or with a long timespan. When the participants use an execution environment, however, the data collection process can be (at least partially) automated. The execution environment becomes responsible for observing the participant, obtaining measurements, storing them, and transferring them to the experimenter.

In a typical software engineering experiment that focuses on the software engineering process, a wide range of data is routinely collected: (1) *time* in various resolutions (e.g., overall, per task, per sub-activity, etc.); (2) detailed *action logs* of the participants; (3) the *results* of executing a task, possibly including intermediate results; and (4) *questionnaire answers* given by the participants. To obtain accurate time measurements, for example, the execution environment can be designed such that it detects suspected idle time, or allows the participant to explicitly pause and resume work.

Collecting as much data as possible via an integrated execution environment has several advantages. First, participants do not need to leave the environment, which reduces their need to switch contexts. Second, it keeps the data collection process maximally hidden from the participants, which increases the realism of the exercise. Finally, all data is collected together, instead of through a wide range of heterogeneous sources (time tracking tools, paper sheets, online forms, etc.), which drastically simplifies subsequent linking and processing of the data.

4) Non-functional requirements: The study execution environment needs to fulfill several non-functional requirements, such as robustness, security and usability.

Robustness. Failures of or hiccups in the study execution environment may cause data loss, data inconsistency or inaccuracy. The tool should therefore be carefully designed for robustness, for example by regularly storing or submitting intermediate results, and/or by logging the actions of participants. Furthermore, attention should be given to the process of deploying updates with bug fixes. A web-based participant tool (e.g., [13], [14]) can be updated much more easily than a stand-alone tool (e.g., [10], [11]), for example.

Security. Participants in a study may want to peek into (or modify) the internals of the tool they (are forced to) use. Also, tinkering with the tool may allow participants to discover information that is not intended for their treatment group, or disrupt the flow of the experiment. When these situations are not prevented, they become serious threats to validity for the study.

Usability. The tool used by the participants should be straightforward to use. Usability problems can negatively impact the validity of the collected data, for instance measurements related to effort, efficiency and productivity, as well as qualitative data related to the participants' satisfaction or preference.

D. Processing and packaging

1) Processing and analysis: In the analysis or processing phase, the researcher will usually explore and analyze the collected data using existing statistical software (e.g., R, SPSS, or SAS). Therefore, the platform should, at a minimum, make it convenient to interoperate with such existing tools, for example by exporting the collected measurements into common formats (e.g., CSV files). The statistical tests and processing steps are dictated by the study design and therefore, the scripts and techniques employed for statistical analysis could be generated and/or exported automatically from the design workbench.

2) Packaging: To ease the verification, validation and replication of the study by other researchers, the existence of a common platform as described so far is an excellent vehicle to package all the information about an experiment (including the goals and hypotheses, experiment design information, and the raw collected data) in one bundle that can be easily shared. The processing and packaging tool has to consider the privacy of the subjects; data may need to be anonymized, usually already well before actual processing, but certainly before distributing and sharing it with the world. A publicly accessible and searchable repository of such experiment designs and data has the potential of becoming an invaluable vehicle for empirical software engineering research.

III. Discussion

Apart from the claimed benefits, there are some potential drawbacks associated to the envisioned platform.

a) Tool impact/bias: The execution environment may have an impact on the tool that is offered to subjects to conduct the experiments, by imposing constraints that limit the tool in such a way that it becomes artificial and no longer representative of the current state of practice. In addition, instrumenting these tools for the purpose of controlling or monitoring the execution of tasks may negatively influence the behavior of these tools, as discussed above.

b) The 'expert effect': Another risk is that while the design tool may provide suggestions or guidelines to an experimental researcher, it will not fully replace the necessity of expertise, know-how and careful experiment design. Including templates of known experiment designs may also hamper creativity, i.e. experimenters may be more inclined to adopt a sub-optimal design for their study, just because it was suggested by the workbench.

IV. Conclusions

In practice, the scientific quality of an empirical study depends highly on the scientific training, domain expertise, honesty and dedication of the experimental researcher that is behind the study. These constraints, together with the observation that software engineering is an inherent human activity that is resource-intensive to investigate, are amongst the many reasons why empirical research in software engineering is still not as widely adopted as it should.

In this paper, we have presented our vision towards an integrated tooling platform for designing, conducting, processing and disseminating empirical software engineering studies. We have argued how such a platform would lower the threshold for setting up experimental studies, as well as strengthen their validity and reproducibility, and in general would contribute to 'better science'.

Acknowledgment

This research is partially funded by the Research Fund KU Leuven and the Secure Design project of the imec HI2 Distributed Trust program.

References

[1] V. R. Basili and M. V. Zelkowitz, "Empirical studies to build a science of computer science," *Communications of the ACM*, vol. 50, no. 11, pp. 33–37, 2007.

[2] A. J. Ko, T. D. Latoza, and M. M. Burnett, "A practical guide to controlled experiments of software engineering tools with human participants," *Empirical Software Engineering*, vol. 20, no. 1, pp. 110–141, 2015.

[11] ——, "Do security patterns really help designers?" in *37th International Conference on Software Engineering (ICSE'15)*, vol. 1, May 2015, pp. 292–302.

[3] S. Hanenberg and A. Stefik, "On the need to define community agreements for controlled experiments with human subjects: a discussion paper," in *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools*. ACM, 2015, pp. 61–67.

[4] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer, 2012.

[5] N. Juristo and A. M. Moreno, *Basics of software engineering experimentation*. Springer, 2013.

[6] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2017. [Online]. Available: https://www.R-project.org

[7] P. Johnson. (2017) Hackystat. [Online]. Available: http://csdl.ics.hawaii.edu/research/hackystat/

[8] M. J. Anderson and P. J. Whitcomb, *DOE simplified: practical tools for effective experimentation*. CRC Press, 2016.

[9] SAS Institute, Inc., "Sas jmp 13," 2017. [Online]. Available: http://www.jmp.com

[10] K. Yskout, R. Scandariato, and W. Joosen, "Does organizing security patterns focus architectural choices?" in *34th International Conference on Software Engineering (ICSE'12)*, June 2012, pp. 617–627.

[12] F. Saleh and M. El-Attar, "A scientific evaluation of the misuse case diagrams visual syntax," *Information and Software Technology*, vol. 66, pp. 73–96, 2015.

[13] M. Riaz, J. Slankas, J. King, and L. Williams, "Using templates to elicit implied security requirements from functional requirements-a controlled experiment," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2014, p. 22.

[14] M. Riaz, J. King, J. Slankas, L. Williams, F. Massacci, C. Quesada-López, and M. Jenkins, "Identifying the implied: Findings from three differentiated replications on the use of security requirements templates," *Empirical Software Engineering*, 2016.

[15] K. Wuyts, R. Scandariato, and W. Joosen, "Empirical evaluation of a privacy-focused threat modeling methodology," *Journal of Systems and Software*, vol. 96, pp. 122–138, 2014.

[16] V. R. Basili, "Software modeling and measurement: The goal/question/metric paradigm," University of Maryland, Tech. Rep. CS-TR-2956, UMIACS-TR-92-96, Sep 1992.

[17] D. C. Montgomery, *Design and analysis of experiments*, 8th ed. Wiley, 2012.

[18] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528 – 540, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X08000861

[19] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for e-Science: Scientific Workflows for Grids*. Springer Publishing Company, Incorporated, 2014.